MICROCHIP

ANXXXX

Advanced Usage of Microchip RN4020

Introduction

Microchip OEM module RN4020 is one of the feature rich Bluetooth Smart solutions in the market. By reading its data sheet and user guide, it is very easy to start a project with basic Bluetooth service/characteristic operations. This application note exposes more detailed, in-depth description of advanced functions of RN4020 module to help users to utilize its full potential.

Microchip Low energy Data Profile

Unlike Serial Port Profile (SPP) in Bluetooth Classic, Bluetooth Low Energy (BTLE) technology does not define a public profile for streaming data as a cable replacement solution. Microchip defines its own proprietary data streaming solution Microchip Low energy Data Profile (MLDP). When using MLDP to stream data between two RN4020 modules, it behaves like a black box working right out of the box. However, if one end of communication is not RN4020, then developers should have more MLDP implementation details.

MLDP Definition

MLDP is implemented with private service defined by Microchip. Details of MLDP private service are shown below:

Name: Microchip MLDP Service
UUID: 00035b03-58e6-07dd-021a-08123a000300

Abstract:
Microchip MLDP provides data exchange mechanism between two BTLE devices.

Summery:

Microchip MLDP uses private service to allow data exchange between two BTLE devices. Microchip MLDP service also provides control to the data exchange for more advanced usage of MLDP.

Service Dependencies:
This service has no dependencies on other GATT-based services.

GATT Requirements:

Sub-Procedure	Server Requirement
Read Characteristic Descriptors	Mandatory
Write	Mandatory
WriteWithoutResponse	Mandatory if MLDPv2 is enabled
Notification	Mandatory if MLDPv2 is enabled
Indication	Mandatory
Write Characteristic Descriptors	Mandatory

Transport Dependencies

Transport	Supported
Classic	false
Low Energy	true

Error Codes
This service does not define any application error codes that are used in Attribute Protocol.

Service Characteristics

<table>
<tr><th>Overview</th><th>Properties</th><th>Security</th><th>Descriptors</th></tr>
<tr>
<td>Name:
MLDP Data
Description:
The MLDP Data characteristic is used to exchange data between two MLDP supported devices. Data is transferred as byte stream through either Write/Indication for MLDP or WriteWithoutResponse/ Notification for MLDPv2
Type:
MLDP_data
Requirement:
Mandatory</td>
<td>
<table>
<tr><th>Property</th><th>Requirement</th></tr>
<tr><td>Read</td><td>Excluded</td></tr>
<tr><td>Write</td><td>Mandatory</td></tr>
<tr><td>WriteWithoutResponse</td><td>Mandatory for MLDPv2</td></tr>
<tr><td>SignedWrite</td><td>Excluded</td></tr>
<tr><td>Notify</td><td>Mandatory for MLDPv2</td></tr>
<tr><td>Indicate</td><td>Mandatory</td></tr>
<tr><td>WritableAuxiliaries</td><td>Excluded</td></tr>
<tr><td>Broadcast</td><td>Excluded</td></tr>
<tr><td>ExtendedProperties</td><td></td></tr>
</table>
</td>
<td>Optional</td>
<td>
<table>
<tr><th>Overview</th><th>Permissions</th></tr>
<tr>
<td>Name:
Client Characteristic Configuration Presentation Format
Type:
org.bluetooth.descriptor.gatt.client_characteristic_configuration
Requirement:
Mandatory</td>
<td>
<table>
<tr><th>Permission</th><th>Requirement</th></tr>
<tr><td>Read</td><td>Mandatory</td></tr>
<tr><td>Write</td><td>Mandatory</td></tr>
</table>
</td>
</tr>
</table>
</td>
</tr>
<tr>
<td>Name:
MLDP Control
Description:
The MLDP Control characteristic is used to control the MLDP data exchange
Type:
MLDP_control</td>
<td>
<table>
<tr><th>Property</th><th>Requirement</th></tr>
<tr><td>Read</td><td>Excluded</td></tr>
<tr><td>Write</td><td>Mandatory</td></tr>
<tr><td>WriteWithoutResponse</td><td>Mandatory for MLDPv2</td></tr>
<tr><td>SignedWrite</td><td>Excluded</td></tr>
<tr><td>Notify</td><td>Excluded</td></tr>
</table>
</td>
<td>Optional</td>
<td>None</td>
</tr>
</table>

Requirement: Mandatory	Indicate	Excluded		
	WritableAuxiliaries	Excluded		
	Broadcast	Excluded		
	ExtendedProperties			

Name: MLDP_data
Assigned Number: 00035b03-58e6-07dd-021a-08123a000301

Abstract:
This characteristic is used to exchange MLDP data between two devices.

Value Fields

Names	Field Requirement	Format	Minimum Value	Maximum Value	Additional Information
Data	Mandatory	uint8	0	255	None

Name: MLDP_control
Assigned Number: 00035b03-58e6-07dd-021a-08123a0003ff

Abstract:
This characteristic is used to control MLDP data streaming under advanced usage.

Value Fields

Names	Field Requirement	Format	Minimum Value	Maximum Value	Additional Information
Control	Mandatory	uint8	0	2	None

Operation of MLDP

Microchip RN4020 acts as server for MLDP private service according to definition described in the previous section.

If MLDP feature is enabled in RN4020, when it connects to another BTLE device, it acts as GATT server and client at the same time. When acting as GATT client, RN4020 performs service discovery process and try to identify if peer device is a server of Microchip MLDP private service. Based on the service discovery process, the MLDP is implemented in two different ways.

If RN4020 discovers that its connection peer is a server of Microchip MLDP private service, then when it sends data to the peer device, it acts as MLDP client and uses either Write or WriteWithoutResponse to the MLDP_data characteristic; when it receives data from the peer device, it acts as MLDP server accepts data by Write or

WriteWithoutResponse to MLDP_data characteristic from the peer device. MLDP data streaming between two RN4020 modules is performing this way.

If RN4020 discovers that its connection peer is not a server of Microchip MLDP private servce, then it expects the peer device to start indication or notification on MLDP_data characteristic. RN4020 acts as MLDP server all the time under this situation. When RN4020 sends data to the peer device, it uses indication or notification; when it receives data from the peer device, it accepts data by Write or WriteWithoutResponse to MLDP_data characteristic from the peer device.

Typically, MLDP feature support between RN4020 and an application on smart phone or tablet is implemented by the second method. After connection is established between RN4020 and a smart device, it is the smart device app's responsibility to start indication or notification on MLDP_data characteristic. The app will use Write or WriteWithoutResponse to send data to RN4020; the app receives data from RN4020 by accepting indication or notification of MLDP_data characteristic on RN4020.

Advanced Features Build on top of MLDP

MLDP is the foundation of advanced features such as remote command and Over The Air (OTA) device firmware update. The advanced features are controlled by MLDP_control characteristic defined in MLDP private service.

After RN4020 is connected with a peer device, it enters remote command mode under following conditions:

1. Remote command support is turned on by setting bit 0x00200000 in the parameter of 'SR' command.
2. Value 0x01 is written to MLDP_control characteristic.

Once RN4020 remote command feature is enabled, it automatically goes into MLDP mode and send ASCII text "RMT_CMD\r\n" to the peer device to indicate the start of remote command mode. From then on, any data received from MLDP_data characteristic will be processed as ASCII command. Any command output will be sent to peer device via MLDP_data characteristic.

When value 0x00 is written to MLDP_control characteristic, RN4020 will immediately exit remote command mode.

After RN4020 is connected with a peer device, it enters OTA mode under following conditions:

1. OTA feature is turned on by setting bit 0x00008000 in parameter of 'SR' command.
2. Value 0x02 is written to MLDP_control characteristic.

Once RN4020 enters OTA mode, it automatically goes into MLDP mode and send ASCII text “OTA\r\n” to the peer device to indicate the start of OTA mode. From then on, any data received from MLDP_data characteristic will be processed as DFU image. If firmware update is successful, RN4020 will automatically reset, otherwise, message “Update Err” will be sent to the peer device via MLDP_data characteristic. Multiple tries of streaming DFU image could be performed. Similar to DFU operation through UART interface, power cycle after failed firmware update likely will brick the module.

To develop an app on a smart device, indication or notification of MLDP_data characteristic on RN4020 needs to be started before writing value MLDP_control characteristic. This procedure is to ensure bi-directional data exchange with RN4020 after getting into remote command or OTA mode.

Low Power Implementation

BTLE targets applications with low power consumption. Microchip puts high priority of low power consumption when developing RN4020. We hope that following information will help RN4020 users to fully use RN4020 low power features.

RN4020 has three main power modes: dormant mode, deep sleep mode and active mode, in the order of ascending power consumption.

Dormant Mode

Dormant mode has the lowest power consumption but has no wireless activity. No BTLE connection can be maintained under this mode. For RN4020 module, waking from dormant mode is the same as rebooting the module. The whole process of waking from dormant mode lasts 1-2 seconds.

To get into dormant mode with lowest current consumption, following procedures need to be followed:

1. Pull BT_WAKE pin low
2. Pull SW_WAKE pin high and get into command mode
3. Issue command ‘O’ to get into Dormant mode
4. If UART is not used, RX pin needs to pull high

To wake up from dormant mode, pull BT_WAKE pin high. After 1-2 seconds, RN4020 will wake up and then be able to wake up from deep sleep mode by pulling SW_WAKE pin. Alternatively, user could pull SW_WAKE pin high before pulling

BT_WAKE pin high, so that once RN4020 wakes up from Dormant mode, it will enter command mode immediately.

Deep Sleep Mode

Deep sleep mode is the most common low power mode that RN4020 may stay in if powered by battery. Under deep sleep mode, RN4020 firmware still periodically wakes up the radio and on board MCU to advertise, maintain BTLE connection or process data. Between the scheduled wireless activities, RN4020 module is put into deep sleep that only consumes a few μA. The total power consumption is then decided by two main factors:

1. Output RF power
2. Advertisement interval or connection parameters

Under deep sleep mode, UART can only operate with baud rate 2400. RN4020 could not guarantee the data integrity under any baud rate other than 2400.

Deep sleep mode is controlled by SW_WAKE pin. A falling edge on SW_WAKE pin put RN4020 module into deep sleep and a rising edge on SW_WAKE pin exit deep sleep mode.

Active Mode

Similar to deep sleep mode, RN4020 firmware periodically powers the transceiver to perform wireless activities to advertise or maintain BTLE connection. The difference of active mode and deep sleep mode is how on board MCU behaves between wireless activities. In active mode, onboard MCU is active all the time; in deep sleep mode, onboard MCU is put into deep sleep. Since MCU is active, higher UART baud rates are supported.

Deep Sleep or Active Mode

Whether to get into active mode to save power is a decision of individual application.

If the application is host-less by either script or remote command, most probably, the lowest power consumption solution is running RN4020 in deep sleep all the time.

If the application has a host MCU, then whether to get into active mode for lowest power consumption depends mostly on data throughput and host MCU power

consumption. For active mode, even though it consumes more power, but support higher UART baud rate. Therefore, if there are a lot of data to send, the whole system could finish sending data over UART in shorter time and both RN4020 module and host MCU could get into deep sleep quicker. On the other hand, under RN4020 deep sleep mode, even though RN4020 consumes less current, but because UART baud rate is lowest at 2400, host MCU must stay awake longer to finish transmit all data over UART, therefore, host MCU consumes more current. For each individual application, whether use active mode could be decided when data throughput and host MCU power consumption are known factors.

Script

RN4020 provides a unique feature that enables users to write their own script and load it to the module without compilation. Making best use of this functionality enables application with lower power consumption, easy development cycle and quick time to market.

Following topics help user to have a better understanding of RN4020 script function.

Script Efficiency

RN4020 runs script by interprets line by line. As the result, the script runs faster if settings or configurations are done in the power on event. For instance, in table 1, two scripts perform the same functionality. However, script from the left defines the variable range for each timer event, while script from the right only defines the variable range in power on event. As the result, for script on the left, RN4020 script interpreter parses and executes the range definition many times when timer event is triggered. However, for script on the right, RN4020 script interpreter only has to parse and execute the range definition once. By choosing right way for script programming, we could gain better efficiency in execution.

Table 1: Script of Different Efficiency

Script of Less Efficiency	Script of More Efficiency
@PW_ON SM,1,00500000 @TM1 $VAR1 < “0123” $VAR1 = @I,0 SM,1,00500000	@PW_ON $VAR1 < “0123” SM,1,00500000 @TM1 $VAR1 = @I,0 SM,1,00500000

Programming Components

Procedure programming has three major components: function, branch and loop. RN4020 script function implements all three major components with its own way.

Function

All RN4020 ASCII commands could be used in script, which serves as predefined function call. In addition, variable could be defined, assigned and used as parameters of ASCII commands, the same way that parameters are used in function calls.

Since version 1.20, RN4020 script engine introduces remote function call feature. Through a private characteristic, this feature allows user to pass multiple input parameters to a script function and retrieve the return value of the function call. In a sense, user could treat a definition of remote function as self-defined event. Over BTLE characteristic access, the remote function call allows user to trigger an event as remote function call.

The remote function feature has following requirements for RN4020 that runs script and its peer device:

- RN4020
    - A private characteristic should be defined with mandatory properties of Write or WriteWithoutResponse. If the remote function call expects return value, then properties such as notification or indication should be supported for automatic result notification; or enable Read property for manual result retrieval. RN4020 acts as GATT server for this private characteristic.
    - A remote function must be defined and such remote function must be associated with the handle of private characteristic mentioned above. A write operation to this private characteristic will trigger the associated remote function call. One or more input parameters could be passed to the remote function, as long as the peer device write the value of private characteristic in predefined format. The remote function then starts to run automatically.
    - Once the remote function call is finished, if there is a return value, such return value needs to be written to the private characteristic.
- Peer Device
    - Peer device should be a client of the defined private characteristic. After connection, peer device should perform service discovery to find the predefined characteristic. If automatic notification for return value of the remote function is preferred, notification or indication needs to start from the peer device before the first remote function call.

- A remote function in the RN4020 script is triggered by a Write or WriteWithoutResponse operation to the associated private characteristic. If no input parameter is expected, a dummy data could be written to the characteristic. Otherwise, input parameters needs to be written in ASCII format and separated by coma. When parameters are passed by ASCII and separated by coma, it is possible to pass more than one parameters in a single write operation to the private characteristic.
- If remote function return value could be automatically retrieved by notification or indication. Optionally, peer device could also use Read the value of private characteristic after it is certain that remote function execution has been finished. If timing is sensitive to receive return value, or it is hard to predict execution time of remote function, it is recommended to use the first method to retrieve the return value of remote function in RN4020 script.

Branch

In procedure programming, branch is implemented by key word "if(condition) else". RN4020 script engine implement branch function by defining variable range. For instance, table 2 shows how branch is implemented in RN4020 script comparing to typical procedure programming.

Table 2: Branch in RN4020 Script

Procedure Programming	RN4020 Script

if((Variable < 0x1234) || (Variable > 0xABCD))
{
  SetVariable(Variable);
}
	$VAR1 < "1234" || $VAR1 > "ABCD"
	SHW,0018,$VAR1

Above mechanism may work in most of the cases, but under more complex branch case, the variable in the condition may not be one of the parameters in function. Table 3 shows an example of such situation.

Table 3: More Complex Branch in Procedure Programming

```
Variable2 = 10;
if( (Varaible1 < 0x1234) || (Varaible1 > 0xABCD) )
{
  SetVaraible(Variable2);
  Advertise();
}
```

To handle such more complex branch, RN4020 script engine provides a work around to bypass it. When interpreting a line in the script, RN4020 script engine performs range checking in following order:

- RN4020 script engine first checks if any variable in the line.
- If a variable is found, then RN4020 script engine continue to check if there is an existing range limit associated with the variable.
- If an existing range limit for the variable is found, such range limit will be checked against the current value of variable.
- If variable value is out of the range, then such script line is ignored.
- If variable value is within the range, the variable is replaced by its current value and then is passed to command parser to execute.

Once the script line is passed to the command parser, the syntax of the script line is checked against the command syntax. If syntax check is passed, the command is executed. However, the syntax check is not performed on trailing data after valid syntax format. As the result, commands are parsed in following way:

- Command "V,1234" is the same as command "V". Since command "V" expect no parameter, so any content after valid syntax "V" is discarded.
- Command "SHW,0018,1234,5678" is performed the same as "SHW,0018,1234".
- Command "A,0100" is not the same as command "A". Because command "A" accepts optional parameters.
- Command "A,0100,1000,1234" is the same as command "A,0100,1000", where "1234" is after all optional parameters.

Combining detailed information of how RN4020 implements its script engine and command parser, we could handle more complex branch problem by adding condition variable as unnecessary command parameter. Table 4 shows how RN4020 script engine handles complex branch.

Table 4 RN4020 Script for Advanced Branch

Procedure Programming	RN4020 Script

Variable2 = 10;
if((Varaible1 < 0x1234) || (Varaible1 > 0xABCD))
{
  SetVaraible(Variable2);
  Advertise();
}
	$VAR2 = "000A"
	$VAR1 < "1234" || $VAR1 > "ABCD"
	SHW,0018,$VAR2,$VAR1
	A,0020,1000,$VAR1

RN4020 could also support branch operation such as "if (condition) else". Table 5 shows an example of such implementation.

Table 5 RN4020 Script for "if(condition) else"

Procedure Programming	RN4020 Script

if((Varaible > 0x1122) && (Varaible < 0x3344))
{
  readAnalogPort(1);
}
else
{
  readAnalogPort(2);
}
	$VAR1 > “1122” && $VAR1 < “3344”
	$VAR2 = @I,1,$VAR1
	$VAR1 < “1123” || $VAR1 > “3343”
	$VAR2 = @I,2,$VAR1

Loop

Another major component of procedure programming is loop, such as key word “for(;;)” or “while(condition)”. RN4020 script engine uses timer to perform loop operation. When RN4020 timer starts, timeouts and then restarts, loop logic is performed. For instance, table 6 shows loop to read analog port.

Table 6 Loop in RN4020 Script

Procedure Programming	RN4020 Script

while(true)
{
  readAnalogPort(1);
}
	@PW_ON
	SM,1,00500000
	
	@TM1
	$VAR1 = @I,1
	SM,1,00500000

This approach works fine for infinite loop, but for loop with more control, we could implement it by using a variable and two timers. Table 7 shows an example of RN4020 script to do finite loop.

Table 7 Finite Loop in RN4020 Script

Procedure Programming	RN4020 Script

for(i = 0; i < 10; i++)
{
  readAnalogPort(1);
}
	@PW_ON
	$VAR2 = “0001” // control var for loop
	$VAR2 > “0000” // must be true to loop
	SM,1,00500000 // time out in 5 sec
	SM,2,03300000 // timeout in 51 sec
	
	@TM1
	$VAR1 = @I,1,$VAR2 // read if control
	// var is true
	SM,1,00500000,$VAR2 // restart timer
	// if control var
	// is true

	@TM2
	$VAR2 = “0000” // after ten loops
	// set control var
	// false

Summery

RN4020 is very powerful when its advanced features are fully utilized. This application note only describes a subset of advanced usage of RN4020. For more information about using RN4020 in creative way, please contact your local Microchip support.